



**Apache Maven Project**  
<http://maven.apache.org/>

## 1. Introduction

Apache Maven est un outil permettant d'automatiser la gestion d'un projet Java. Il offre entre autres les fonctionnalités suivantes :

- Compilation et déploiement des applications Java (JAR, WAR)
- Gestion des librairies requises par l'application en utilisant des dépôts (*repository*) local ou distant
- Exécution des tests unitaires
- Génération des documentations du projet (site web, pdf, Latex)
- Intégration dans différents IDE (Eclipse, NetBeans, etc.)

Ce tutoriel va vous apprendre à utiliser Maven dans tout projet de développement utilisant Java. Après avoir terminé ce tutoriel, vous vous retrouvez avec un niveau d'expertise modéré dans l'utilisation d'Apache Maven.

Pour résumer, Maven simplifie et standardise le processus de construction d'un projet Java. Il gère la compilation, la distribution, la documentation, la collaboration en équipe et d'autres tâches de façon transparente. Maven augmente la réutilisabilité et prend en charge la plupart des tâches liées à la construction.

## 2. Prérequis

- Java : JDK 1.7 ou plus récent.
- RAM : Pas de minimum requis
- Espace disque :
  - Environ 10 Mo sont nécessaires pour l'installation de Maven.
  - Un espace disque supplémentaire sera utilisé pour le dépôt local de Maven dont La taille varie en fonction de l'utilisation, mais il faut prévoir au moins 500 Mo.
- Système d'exploitation : Aucun prérequis.
  - Disponible sous Windows, Linux, MAC, etc...

### 3. Installation d'Apache Maven

#### Etape 1: Vérifier l'installation de Java sur votre machine

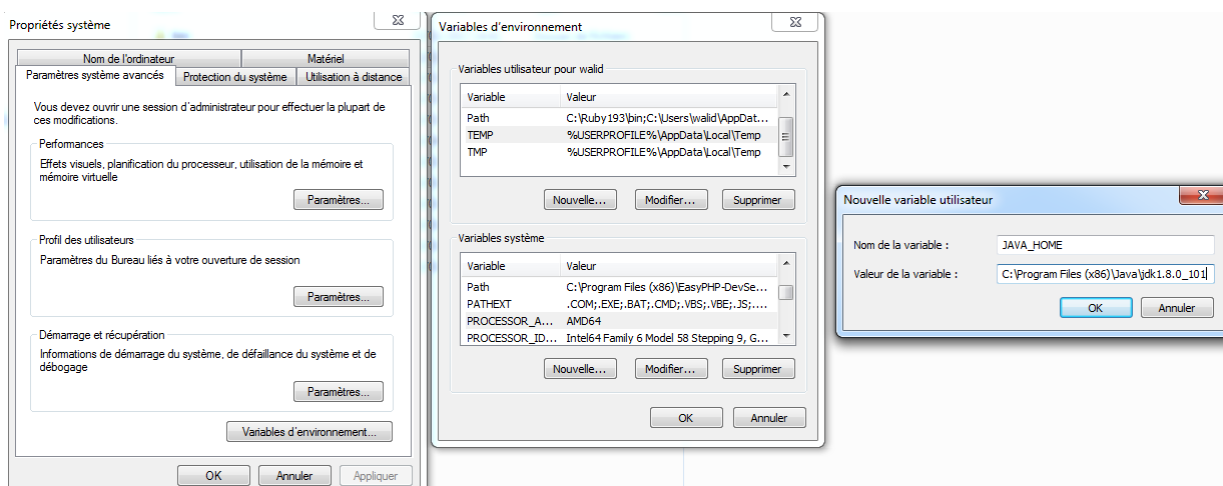
```
C:\Users\walid>java -version
java version "1.8.0_151"
Java(TM) SE Runtime Environment (build 1.8.0_151-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.151-b12, mixed mode)
```

Si Java n'est pas installé, téléchargez et installez le Kit de Développement Java (SDK) depuis le site officiel :

<https://www.oracle.com/technetwork/java/javase/downloads/index.html>.

#### Etape 2: Définir l'environnement JAVA

Définissez la variable d'environnement **JAVA\_HOME** pour qu'elle pointe vers l'emplacement du répertoire de base où Java est installée sur votre machine. Par exemple :



#### Etape 3: Télécharger l'Archive Maven

- Télécharger Maven depuis son site web officiel <https://maven.apache.org/download.cgi>
- Pour Windows : <http://www-us.apache.org/dist/maven/maven-3/3.5.3/binaries/apache-maven-3.5.3-bin.zip>



## Etape 4: Extraire l'archive de Maven

Extraire l'archive dans le répertoire dans lequel vous souhaitez installer Maven 3.5.3. Le sous-répertoire apache-maven-3.5.3 sera créé à partir de l'archive. Par exemple : C:\Program Files (x86)\apache-maven-3.5.3

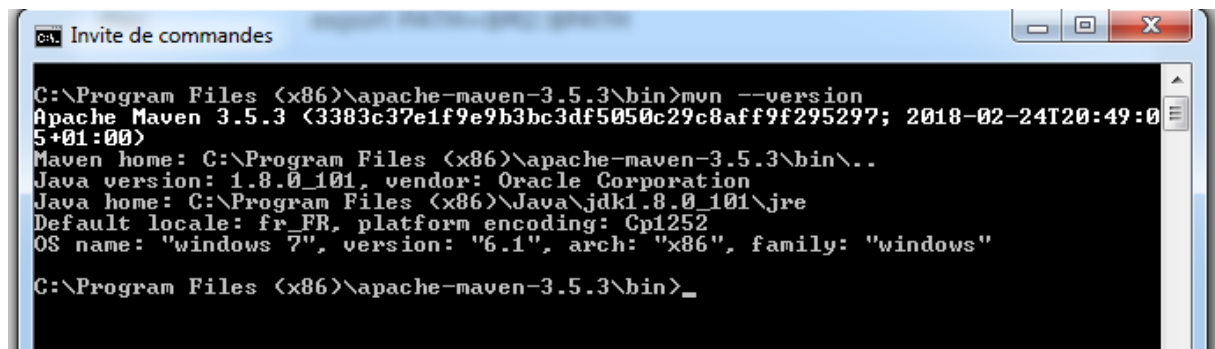
## Etape 5: Définir les variables d'environnement Maven

- Ajoutez M2\_HOME, M2, MAVEN\_OPTS aux variables d'environnement.
  - SET M2\_HOME C:\Program Files (x86)\apache-maven-3.5.3
  - SET M2 %M2\_HOME%\bin
  - SET MAVEN\_OPTS -Xms256m -Xmx512m

## Etape 6: Ajouter l'emplacement du répertoire bin du Maven au chemin du système

- Ajoutez la variable M2 au chemin du système (à la variable PATH).

## Etape 7: Vérifier l'installation



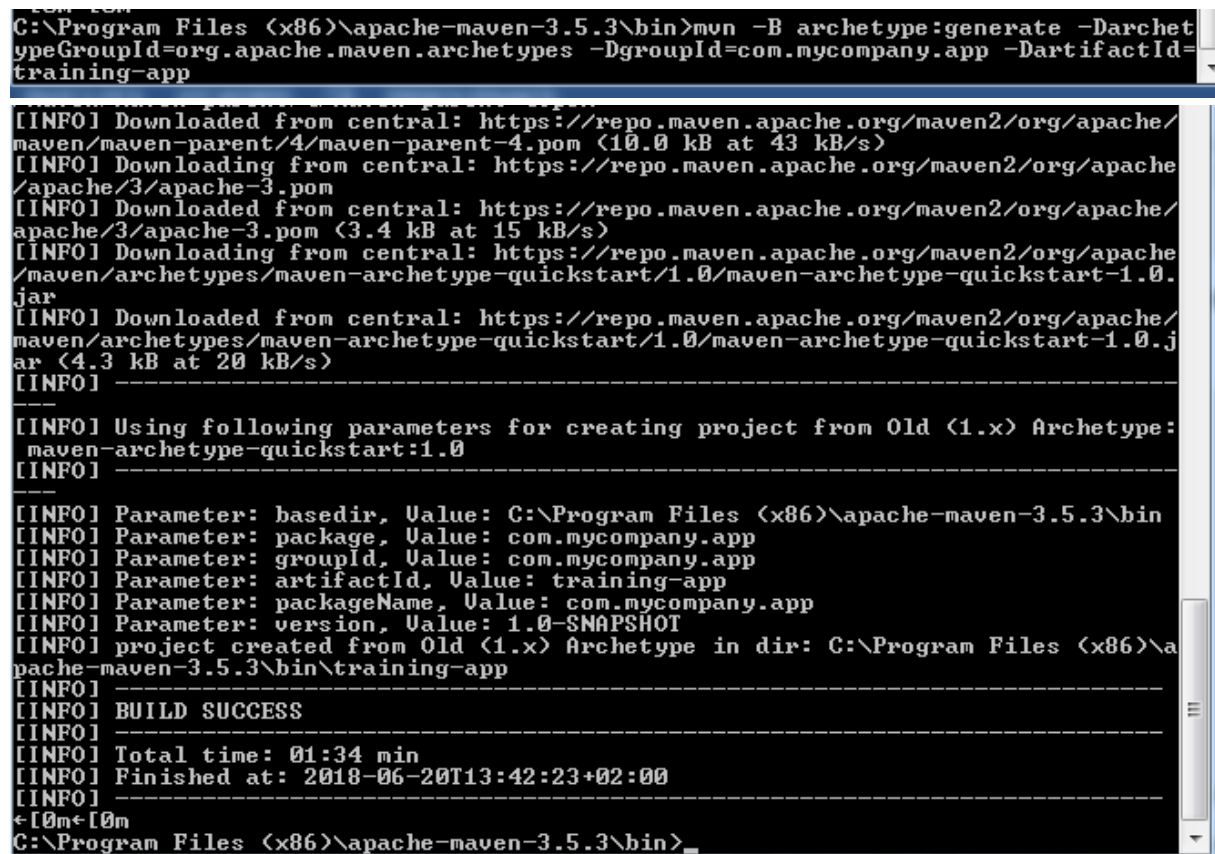
- Attendre la fin du démarrage du serveur qui sera signalée par l'affichage de la ligne suivante:

## 4. Mon premier projet Maven

On veut créer notre premier projet Maven. Pour cela, nous allons utiliser le mécanisme d'Archetype de Maven. Un Archetype est défini comme un modèle à partir duquel notre projet sera créé. Dans Maven, un Archetype pourrait être combiné avec une entrée de l'utilisateur pour produire un projet adapté aux besoins de l'utilisateur.

Pour créer un nouveau projet Maven, exécutez ce qui suit à partir de la ligne de commande:

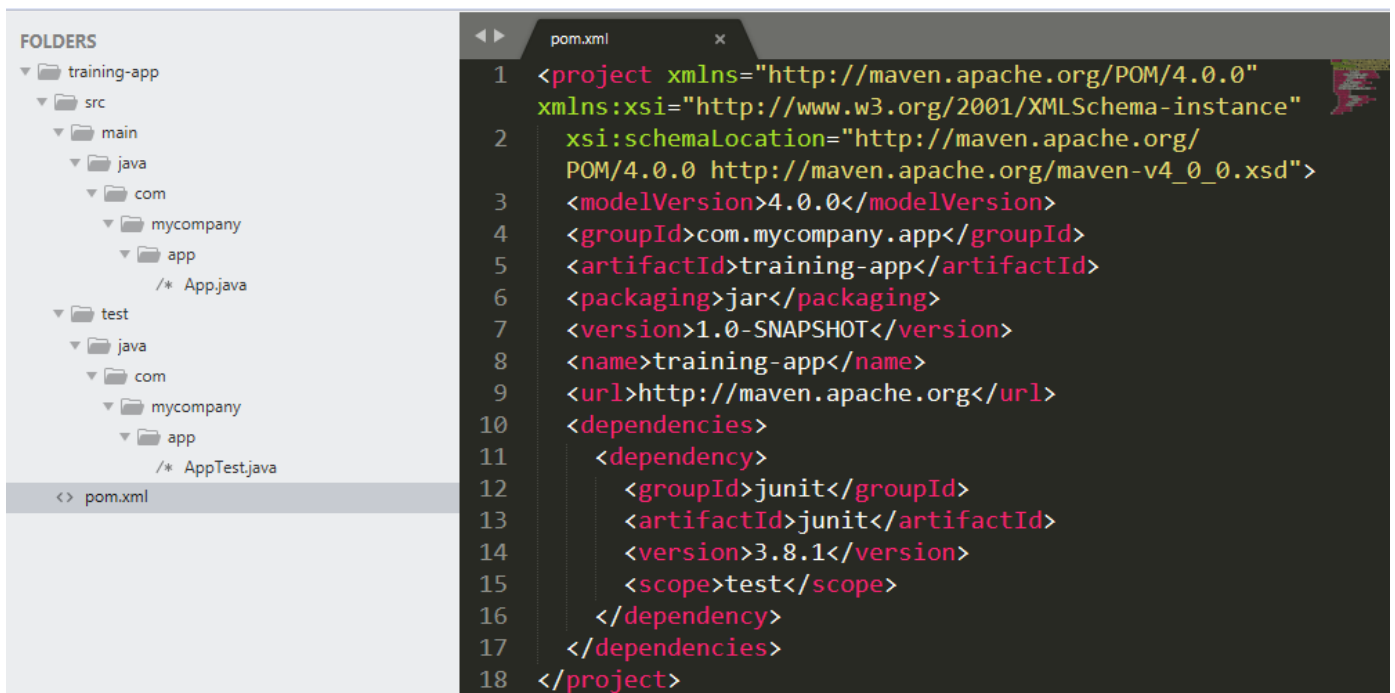
```
$ mvn -B archetype:generate \
-DarchetypeGroupId=org.apache.maven.archetypes \
-DgroupId=com.mycompany.app \
-DartifactId=training-app
```



```
C:\Program Files (x86)\apache-maven-3.5.3\bin>mvn -B archetype:generate -DarchetypeGroupId=org.apache.maven.archetypes -DgroupId=com.mycompany.app -DartifactId=training-app

[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-parent/4/maven-parent-4.pom (10.0 kB at 43 kB/s)
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/apache/3/apache-3.pom
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/apache/3/apache-3.pom (3.4 kB at 15 kB/s)
[INFO] Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar
[INFO] Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.0/maven-archetype-quickstart-1.0.jar (4.3 kB at 20 kB/s)
[INFO] -----
[INFO] Using following parameters for creating project from Old (1.x) Archetype:
maven-archetype-quickstart:1.0
[INFO] -----
[INFO] Parameter: basedir, Value: C:\Program Files (x86)\apache-maven-3.5.3\bin
[INFO] Parameter: package, Value: com.mycompany.app
[INFO] Parameter: groupId, Value: com.mycompany.app
[INFO] Parameter: artifactId, Value: training-app
[INFO] Parameter: packageName, Value: com.mycompany.app
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Program Files (x86)\apache-maven-3.5.3\bin\training-app
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:34 min
[INFO] Finished at: 2018-06-20T13:42:23+02:00
[INFO] -----
C:\Program Files (x86)\apache-maven-3.5.3\bin>
```

Une fois que vous avez exécuté cette commande, vous remarquerez que certaines choses se sont produites. Tout d'abord, vous remarquerez qu'un répertoire nommé training-app a été créé pour le nouveau projet, et ce répertoire contient un fichier nommé pom.xml. La structure du projet devrait ressembler à ceci:



Comme vous pouvez le voir, le projet créé à partir de l' d'Archetype a un dossier *main* pour les sources de votre application et un dossier *test* pour vos sources de test. C'est la disposition standard pour les projets Maven (les sources d'application résident dans `${basedir}/src/main/java` et les sources de test résident dans `${basedir}/src/test/java`, où `${basedir}` représente le répertoire contenant le fichier `pom.xml`).

Aussi, le projet créé possède un POM. Le fichier ***pom.xml*** contient le modèle d'objet de projet (POM Project Object Model) pour ce projet. Maven est centré autour de la notion de projet, et le POM présente l'unité de travail de base. En bref, le POM contient toutes les informations importantes sur votre projet.

Le fichier ***pom.xml*** s'agit d'un POM très simple, et qui affiche toujours les éléments clés de chaque POM. Parcourons chacun d'entre eux pour vous familiariser avec les éléments essentiels de POM:

Éléments POM	Description
<b>project</b>	C'est l'élément de premier niveau dans tous les fichiers <code>pom.xml</code> Maven.
<b>modelVersion</b>	Cet élément indique quelle version du modèle d'objet utilise ce POM. La version du modèle elle-même change très rarement (4.0.0 est la version actuelle)
<b>groupId</b>	Cet élément indique l'identifiant unique de l'organisation ou du groupe qui a créé le projet. Le <code>groupId</code> est l'un des identifiants de clé d'un projet et est généralement basé sur le nom de domaine complet de votre organisation. Par exemple <code>org.apache.maven.plugins</code> est le <code>groupId</code> désigné pour tous les plugins Maven.

<b>artifactId</b>	Cet élément indique le nom de base unique de l'artefact primaire généré par ce projet. L'artefact principal d'un projet est généralement un fichier JAR. Les artefacts secondaires tels que les bundles sources utilisent également l'artifactId comme partie de leur nom final. Un artefact typique produit par Maven aurait la forme <artifactId> - <version>. <Extension> (par exemple, trainingapp-1.0.jar).
<b>packaging</b>	Cet élément indique le type de package à utiliser par cet artefact (par exemple, JAR, WAR, EAR, etc.). Cela ne signifie pas seulement que l'artefact produit est JAR, WAR ou EAR mais peut également indiquer un cycle de vie spécifique à utiliser dans le cadre du processus de construction. La valeur par défaut de l'élément de packaging est JAR. Vous n'avez donc pas besoin de spécifier cela pour la plupart des projets.
<b>version</b>	Cet élément indique la version de l'artefact généré par le projet. La gestion des versions dans Maven est basée sur le concept de SNAPSHOT. Dans une version, un SNAPSHOT indique qu'un projet est en cours de développement.
<b>name</b>	Cet élément indique le nom d'affichage utilisé pour le projet. Ceci est souvent utilisé dans la documentation générée par Maven.
<b>url</b>	Cet élément indique où le site du projet peut être trouvé. Ceci est souvent utilisé dans la documentation générée par Maven.
<b>description</b>	Cet élément fournit une description de base de votre projet. Ceci est souvent utilisé dans la documentation générée par Maven.

## 5. Cycle de vie Maven (Build Life Cycle)

### 5.1. Cycle de vie (Définition)

Maven est basé sur le concept de base d'un cycle de vie de construction de projets. Cela signifie que le processus de construction, de distribution et de déploiement d'un projet (artefact particulier) est clairement défini.

La construction d'un projet est défini par un ensemble de commandes, et le POM s'assurera d'obtenir les résultats souhaités.

Il existe **trois cycles de vie** de construction intégrés: *default*, *clean* et *site*.

- Le cycle de vie *default* (*Build*) gère le déploiement de votre projet
- le cycle de vie *clean* gère le nettoyage du projet
- le cycle de vie du *site* gère la création de la documentation du site de votre projet.

## 5.2. Les Phases

Un cycle de vie de construction (Build Life Cycle) est **une séquence de phases** (appelées aussi stages) qui définit l'ordre dans lequel les **objectifs** (*goals*) qui doivent être exécutés. Ici, la phase représente une étape du cycle de vie. À titre d'exemple, un cycle de vie Maven Build typique (càd *default*) comprend la séquence de phases suivante :

- **Validate** : Valide que le projet est correctement défini
- **Compile** : Compile les sources
- **Test** : Lance les tests unitaires
- **Package** : Prépare la distribution du projet. (archives Jar, War, Ear...)
- **integration-test** : Lance les tests d'intégration
- **verify** : Lance des tests de validation du package créé.
- **Install** : Installe le package **en local** sur la machine pour pouvoir être réutilisé comme dépendance.
- **Deploy** : Déploie le package **sur un serveur** pour qu'il puisse être **réutilisé** par tout le monde.

## 5.3. Les Objectifs (Goals)

On fournit à Maven 2 une liste de goals à exécuter. Un goal est **une tâche** précise que Maven est en mesure de réaliser à partir des informations qu'il pourra trouver dans le fichier pom.xml.

- Tous les goals se trouvent dans **des plugins** Maven.
- Pour exécuter un goal, Maven va donc commencer par résoudre le nom du goal pour en déduire le plugin dans lequel il se trouve et le télécharger.
- Commande : `mvn <nom du plugin>:<nom du goal>` ou bien directement `mvn <nom du goal>`

## 5.4. Les plug-ins Maven

Maven est en fait un framework d'exécution de plugin où chaque tâche est en fait effectuée par des plugins. Les Plugins Maven sont généralement utilisés pour :

- **Jar** : créer un fichier jar depuis le projet
- **War** : créer un fichier war depuis le projet
- **Compiler** : compiler les fichiers JAVA
- **surefire** : lancer des tests unitaires avec JUnit et créer les rapports de test
- **javadoc** : créer la documentation du projet
- **antrun** : Exécute un ensemble de tâches ant à partir de n'importe quelle phase mentionnée dans la construction (cad dans le fichier pom.xml).



## 6. Manipulations Maven

### 6.1. Construction ET Test (Build & Test)

Ce que nous avons appris dans la section précédente est comment créer une application Java avec Maven et la notion du cycle de vie. Nous verrons maintenant comment lancer les différentes phases d'un cycle de vie afin de tester notre application créée dans la section 4.

#### 6.1.1. Compilation (Build)

Lancer la commande **mvn** avec l'objectif **compile**.

```
Downloaded from central: https://repo.maven.apache.org/maven2/commons-logging/commons-logging-api/1.1/commons-logging-api-1.1.jar <45 kB at 10 kB/s>
Downloaded from central: https://repo.maven.apache.org/maven2/junit/junit/3.8.2/junit-3.8.2.jar <121 kB at 26 kB/s>
Downloaded from central: https://repo.maven.apache.org/maven2/log4j/log4j/1.2.12/log4j-1.2.12.jar <358 kB at 73 kB/s>
Downloaded from central: https://repo.maven.apache.org/maven2/com/google/collections/google-collections/1.0/google-collections-1.0.jar <640 kB at 123 kB/s>
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!
[INFO] Compiling 1 source file to C:\Maven\training-app\target\classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 50.371 s
[INFO] Finished at: 2018-06-21T10:21:17+02:00
[INFO] -----
C:\Maven\training-app>mvn compile
```

- Les classes compilées ont été placées dans `${basedir}/target/classes`, ce qui est une autre convention standard de Maven.
- On remarque que Maven a exécuté dans l'ordre la phase de validation (récupération des ressources et téléchargement des plugins nécessaires) et de compilation sans passer automatiquement aux phases suivantes (*test*, *package*, *install*, *deploy*, etc.) du cycle de vie.

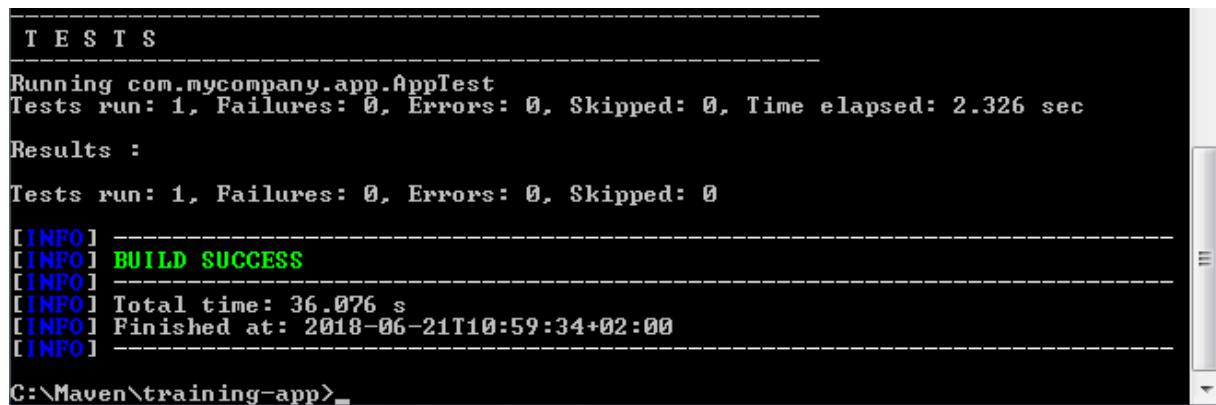
On veut maintenant tester le fichier JAVA compilé :

```
C:\Maven\training-app\target\classes>java com.mycompany.app.App
Hello World!
C:\Maven\training-app\target\classes>
```

### 6.1.2. Tests Unitaires (Unit Test)

Dans le fichier pom.xml, Maven a déjà ajouté le plugin *Junit* comme outil de test unitaire. Dans notre projet, Maven ajoute par défaut un fichier source *App.java* et un fichier de test *AppTest.java*, comme indiqué dans la section 4.

On vous demande maintenant de la lancer la commande ***mvn test*** pour lancer les tests unitaires.



```

-----
T E S T S
-----
Running com.mycompany.app.AppTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.326 sec
Results :
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 36.076 s
[INFO] Finished at: 2018-06-21T10:59:34+02:00
[INFO] -----
C:\Maven\training-app>_

```

A partir des traces d'exécution, quelles étapes ont été exécutées par Maven ?

- Maven télécharge plus de dépendances cette fois-ci. Ce sont les dépendances et les plugins nécessaires à l'exécution des tests (il a déjà les dépendances dont il a besoin pour la compilation et ne les téléchargera plus).
- Avant de compiler et d'exécuter les tests, Maven compile le code principal (pas de changement ici).
- Le rapport de Test reports est disponible dans le dossier \${basedir}/target/surefire-reports
- Les fichiers de test compilés se trouve dans le dossier \${basedir}/target/test-classes
- Pour compiler uniquement les fichiers test unitaire et sans les exécuter, on lance ***mvn test-compile***

### 6.1.3. Génération du fichier JAR

Dans le fichier POM pom.xml, le packaging par défaut est configuré à jar. C'est ainsi que Maven sait produire un fichier JAR à partir de la commande ***mvn package***.

On vous demande maintenant de la lancer la commande ***mvn package*** pour créer une archive jar du notre projet.

```
[INFO] Building jar: C:\Maven\training-app\target\training-app-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 28.388 s
[INFO] Finished at: 2018-06-21T11:14:57+02:00
[INFO] -----
C:\Maven\training-app>mvn package
```

- Le fichier jar est généré dans le répertoire `${basedir}/target`
- On peut mettre le fichier jar dans un repository local (dans `${user.home}/.m2/repository`) ou bien dans un serveur distant.

Essayer d'exécuter le fichier jar avec la commande suivante :

***java -jar target\training-app-1.0-SNAPSHOT.jar***

```
C:\Maven\training-app>java -jar target\training-app-1.0-SNAPSHOT.jar
aucun attribut manifest principal dans target\training-app-1.0-SNAPSHOT.jar
```

On remarque ici que notre fichier jar n'est pas exécutable. En effet, le fait de préciser `<packaging>jar</packaging>` n'indique pas que Maven va créer un jar exécutable avec toutes ses dépendances. Ce jar ne contiendra que les classes compilées de notre projet et ne sera pas exécutable. Pour créer un fichier exécutable, on doit ajouter le plugin ***maven-jar-plugin*** à notre projet.

Ajouter la section suivante dans votre fichier POM :

```
<build>
  <plugins>
    <plugin>
      <!-- Build an executable JAR -->
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-jar-plugin</artifactId>
      <version>3.0.2</version>
```

```

<configuration>

  <archive>

    <manifest>

      <addClasspath>true</addClasspath>

      <classpathPrefix>lib/</classpathPrefix>

      <mainClass>com.mycompany.app.App</mainClass>

    </manifest>

  </archive>

</configuration>

</plugin>

</plugins>

</build>

```

On indique donc ici la classe main (<mainClass>), que notre jar sera créé lors de la phase « packaging » de Maven

Exécuter de nouveau la commande **mvn package** et lancer le fichier jar.

```

[INFO] Building jar: C:\Maven\training-app\target\training-app-1.0-SNAPSHOT.jar
[INFO] BUILD SUCCESS
[INFO] Total time: 16.528 s
[INFO] Finished at: 2018-06-21T11:48:00+02:00
[INFO] -----
C:\Maven\training-app>java -jar target\training-app-1.0-SNAPSHOT.jar
Hello World!
C:\Maven\training-app>

```

### 6.1.4. La commande Clean

On veut maintenant supprimer la compilation actuelle. Exécuter la commande **mvn clean**

```

C:\Maven\training-app>mvn clean
[INFO] Scanning for projects...
[INFO] -----< com.mycompany.app:training-app >-----
[INFO] Building training-app 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ training-app ---
[INFO] Deleting C:\Maven\training-app\target
[INFO] BUILD SUCCESS
[INFO] Total time: 0.540 s
[INFO] Finished at: 2018-06-21T12:15:09+02:00
[INFO] -----
C:\Maven\training-app>

```

○

### 6.1.5. La commande Install

Dans un environnement de développement, on utilise la commande ***mvn install*** pour générer et installer les artefacts **dans le référentiel (repository) local**.

- Cette commande exécute chaque phase du cycle de vie par défaut dans l'ordre (validation, compilation, package, etc.) avant d'exécuter l'installation.
- On aura besoin uniquement d'appeler la dernière phase de construction à exécuter, dans ce cas, lancez:

```
[INFO] Installing C:\Maven\training-app\target\training-app-1.0-SNAPSHOT.jar to
C:\Users\walid\.m2\repository\com\mycompany\app\training-app\1.0-SNAPSHOT\traini
ng-app-1.0-SNAPSHOT.jar
[INFO] Installing C:\Maven\training-app\pom.xml to C:\Users\walid\.m2\repository
\com\mycompany\app\training-app\1.0-SNAPSHOT\training-app-1.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 24.497 s
[INFO] Finished at: 2018-06-21T12:25:11+02:00
[INFO] -----
C:\Maven\training-app>
```

Vérifier que le jar est bien installé dans votre local repository (`${user.home}/.m2/repository`).

### 6.1.6. La commande Deploy

Dans un environnement de build (ou bien production), on utilise la commande ***mvn deploy*** pour créer et déployer proprement les artefacts du notre projet dans un référentiel partagé (shared repository). Ceci représente les cas usuel dans les entreprises.

Pour exécuter un la commande ***mvn deploy*** il faut configurer un serveur repository (voir plus tard).

### 6.1.7. Générer la documentation du projet

La génération de site du projet est l'une des fonctionnalités les plus prisées de Maven. Le POM a assez d'informations pour générer un site web pour votre projet. On peut aussi personnaliser notre site Maven, sinon on peut se limiter à la version générée par défaut en utilisant la commande ***mvn site***.

```

[INFO] Rendering site with org.apache.maven.skins:maven-default-skin:jar:1.0 ski
n.
[INFO] Generating "Dependencies" report --- maven-project-info-reports-plugin
:2.9
[INFO] Generating "Dependency Convergence" report --- maven-project-info-repo
rts-plugin:2.9
[INFO] Generating "Dependency Information" report --- maven-project-info-repo
rts-plugin:2.9
[INFO] Generating "About" report --- maven-project-info-reports-plugin:2.9
[INFO] Generating "Plugin Management" report --- maven-project-info-reports-p
lugin:2.9
[INFO] Generating "Plugins" report --- maven-project-info-reports-plugin:2.9
[INFO] Generating "Summary" report --- maven-project-info-reports-plugin:2.9
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 01:46 min
[INFO] Finished at: 2018-06-21T12:48:34+02:00
[INFO] -----
C:\Maven\training-app>

```

### 6.1.8. Modification des fichiers sources

## 6.2. Gestion de dépendances du projet

Pour en revenir aux principes de Maven :

- Le répertoire *src* ne doit contenir que des fichiers sources apportés au projet
- Les bibliothèques externes (appelées dépendances) utilisées par le projet ne doivent être que des liens vers d'autres artefacts Maven et surtout pas copiés dans le répertoire *src* du projet.
- Un grand nombre d'artefacts jars est disponible sur les entrepôts officiels de Maven : <http://www.mvnrepository.com/>

Maven propose de définir toutes les dépendances par configuration dans le fichier *pom.xml*. C'est ensuite le plugin Maven de gestion de dépendances qui ira télécharger sur les repositories distants les fichiers jar indiqués comme dépendances, s'ils ne se trouvent pas dans le repository local.

On veut maintenant modifier notre projet afin de gérer une liste des sessions de formations. Les informations qu'on voudrait afficher sont stockées dans une base de données MySQL (dans la table *session*). Pour cela, faites les modifications suivantes :

- Modifier la classe *App.java* (le remplacer par le contenu du fichier *Training\_Data\_Service.java*)
- Installer un serveur XAMP, accéder au portail PhpMyAdmin (<http://localhost/phpmyadmin>) et créer une nouvelle base données training
- Importer dans PhpMyAdmin le fichier *training.sql* qui permet de créer la table *Session* (contient deux enregistrements par défaut).
- Ajouter la dépendance *mysql-connector-java* dans le fichier *pom.xml*

```

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <version>5.1.35</version>
</dependency>

```
- Compiler votre projet et tester le fichier jar *mvn clean install && java -jar target\training-app-1.0-SNAPSHOT.jar*

- Faire les modifications nécessaires dans votre fichier POM afin de rendre le plugin *mysql-connector-java* disponible à l'exécution du fichier jar.

**Première méthode** : copier la dépendance dans le dossier lib au moment de compilation du votre projet (dans la phase package cad lors de la génération du jar).

```
<plugin>

  <groupId>org.apache.maven.plugins</groupId>

  <artifactId>maven-dependency-plugin</artifactId>

  <executions>

    <execution>

      <id>copy</id>

      <phase>package</phase>

      <goals>

        <goal>copy-dependencies</goal>

      </goals>

      <configuration>

        <outputDirectory>

          ${project.build.directory}/lib

        </outputDirectory>

      </configuration>

    </execution>

  </executions>

</plugin>

<plugin>
```

```
C:\Maven\training-app>java -jar target\training-app-1.0-SNAPSHOT.jar
----- Connexion au serveur de données MySQL -----
Le driver JDBC pour MySQL est disponible.
Connexion à la base de données a été établie avec succès.
----- Afficher toutes les sessions de formations -----
Formation Integration Continue, Maven, Toulouse, 2018-06-25, 10, 1
Formation Integration Continue, Jenkins, Toulouse, 2018-06-27, 10, 1
```

**Deuxième méthode** : Assembler le fichier jar de la dépendance avec le fichier jar final de l'application en utilisant le plugin ***maven-assembly-plugin***

```
<plugin>

  <artifactId>maven-assembly-plugin</artifactId>

  <configuration>

    <archive>

      <manifest>

        <mainClass>com.mycompany.app.App</mainClass>

      </manifest>

    </archive>

    <descriptorRefs>

      <descriptorRef>jar-with-dependencies</descriptorRef>

    </descriptorRefs>

  </configuration>

  <executions>

    <execution>

      <id>make-assembly</id> <!-- this is used for inheritance merges -->

      <phase>package</phase> <!-- bind to the packaging phase -->

      <goals>

        <goal>single</goal>

      </goals>

    </execution>

  </executions>

</plugin>
```



```

[INFO] Installing C:\Maven\training-app\target\training-app-1.0-SNAPSHOT.jar to
C:\Users\walid\.m2\repository\com\mycompany\app\training-app\1.0-SNAPSHOT\traini
ng-app-1.0-SNAPSHOT.jar
[INFO] Installing C:\Maven\training-app\pom.xml to C:\Users\walid\.m2\repository
\com\mycompany\app\training-app\1.0-SNAPSHOT\training-app-1.0-SNAPSHOT.pom
[INFO] Installing C:\Maven\training-app\target\training-app-1.0-SNAPSHOT-jar-wit
h-dependencies.jar to C:\Users\walid\.m2\repository\com\mycompany\app\training-a
pp\1.0-SNAPSHOT\training-app-1.0-SNAPSHOT-jar-with-dependencies.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.722 s
[INFO] Finished at: 2018-06-21T17:50:44+02:00
[INFO] -----

C:\Maven\training-app>copy target\training-app-1.0-SNAPSHOT-jar-with-dependencie
s.jar ..
        1 fichier(s) copi  (s).

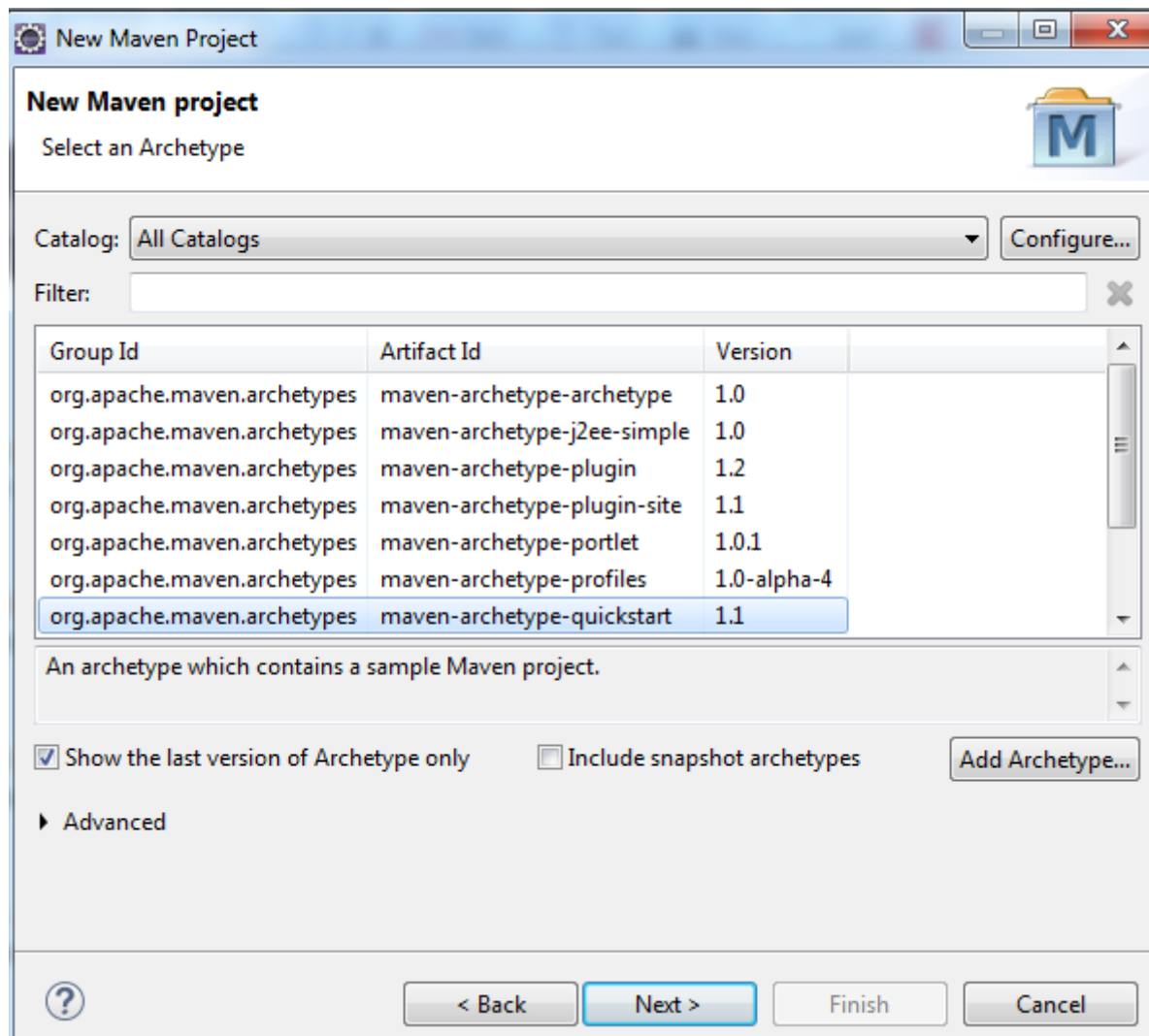
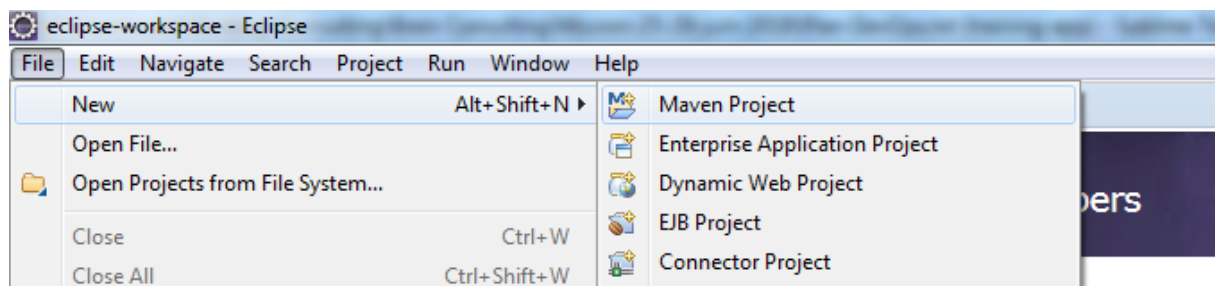
C:\Maven\training-app>cd ..

C:\Maven>java -jar training-app-1.0-SNAPSHOT-jar-with-dependencies.jar
----- Connexion au serveur de donn  es MYSQL -----
Le driver JDBC pour MySQL est disponible.
Connexion    la base de donn  es a   t       tablie avec succ   s.
----- Afficher toutes les sessions de formations -----
Formation Integration Continue, Maven, Toulouse, 2018-06-25, 10, 1
Formation Integration Continue, Jenkins, Toulouse, 2018-06-27, 10, 1

C:\Maven>

```

## 7. Maven Eclipse IDE



**New Maven Project**

Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

Properties available from archetype:

Name	Value

► Advanced

eclipse-workspace - training-app/src/main/java/com/mycompany/app/training\_app/App.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

Project Explorer

- training-app
  - src/main/java
  - src/test/java
  - JRE System Library [J2SE-1.5]
  - Maven Dependencies
  - src
    - main
      - java
        - com
          - mycompany
            - app
              - training\_app
                - App.java
  - test
  - target
  - pom.xml

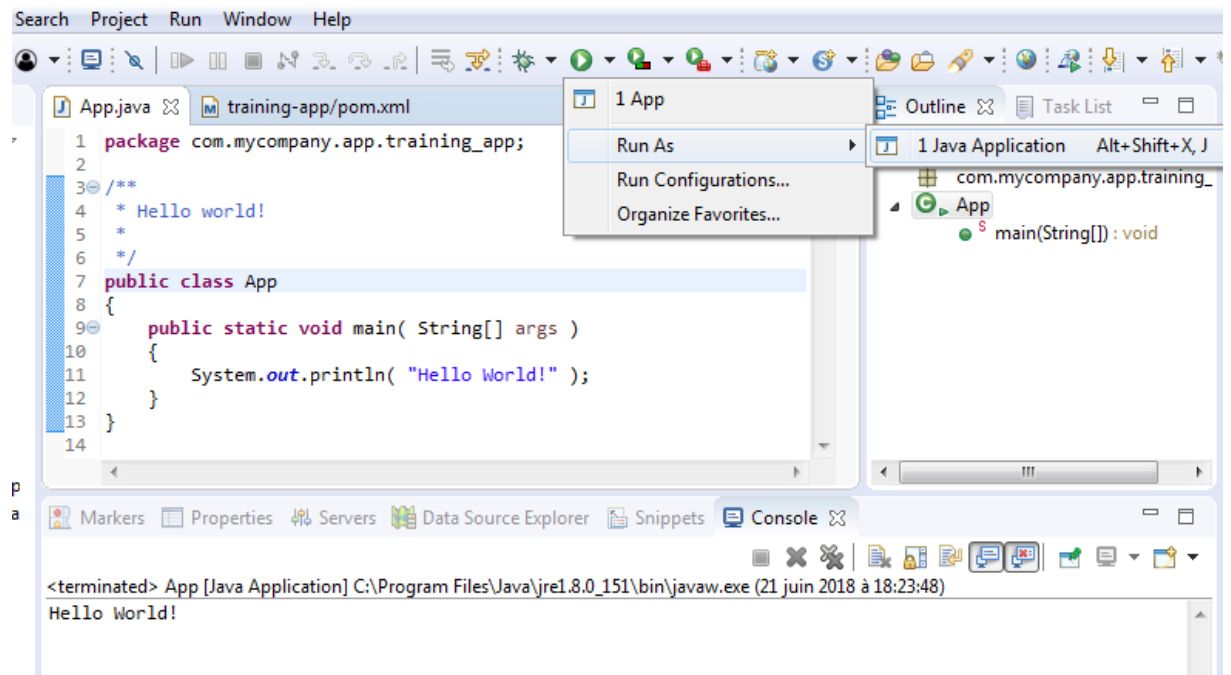
App.java

```

1 package com.mycompany.app.training_app;
2
3 /**
4  * Hello world!
5  *
6  */
7 public class App
8 {
9     public static void main( String[] args )
10    {
11        System.out.println( "Hello World!" );
12    }
13 }
14

```

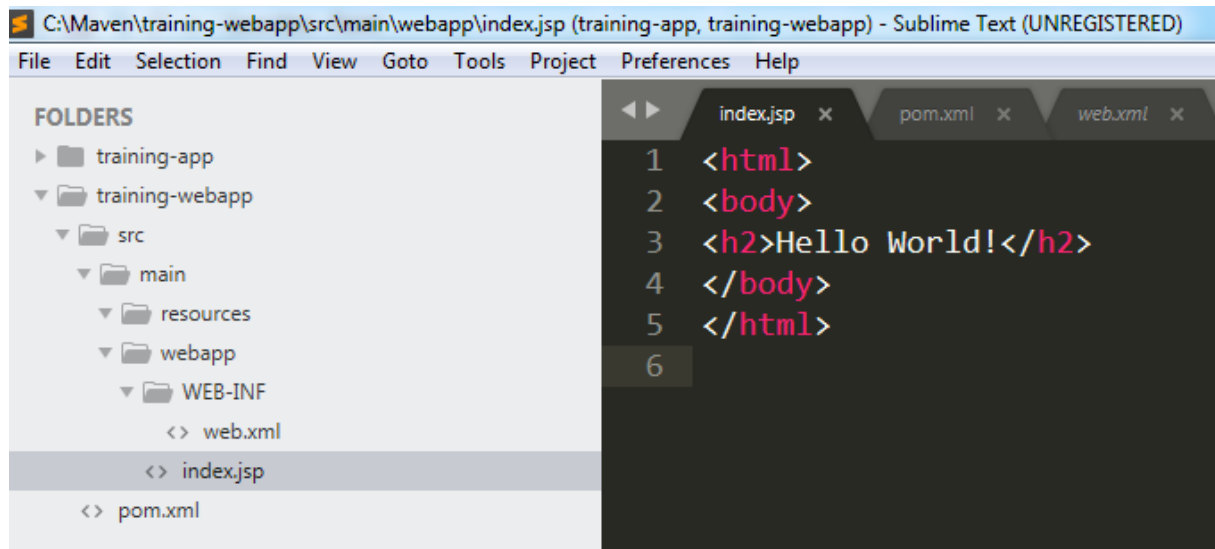
training-app/pom.xml



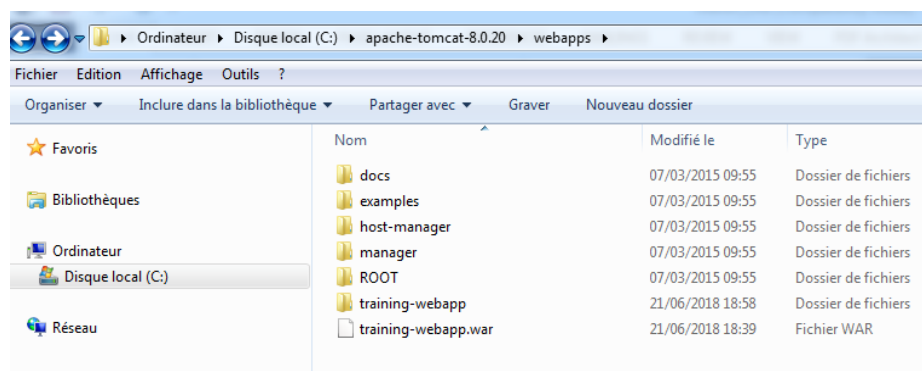
## 8. Application Web

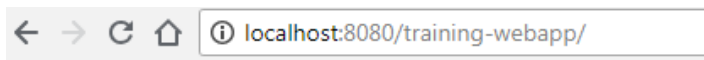
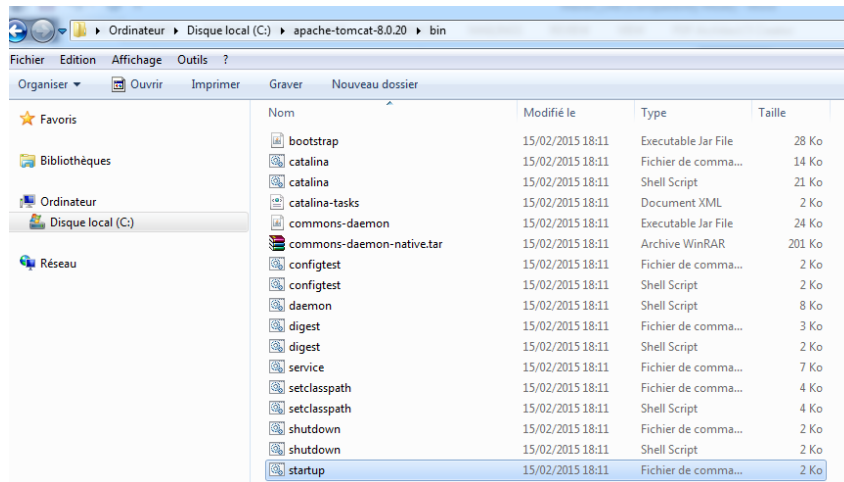
Pour créer une application web java simple, nous allons utiliser le plugin **maven-archetype-webapp**.

```
$ mvn -B archetype:generate \
-DarchetypeArtifactId=maven-archetype-webapp \
-DgroupId=com.mycompany.app \
-DartifactId=training-webapp
```



Compiler et déployer (*mvn install*) votre application en copiant le fichier *war* dans le dossier *webapps* du votre serveur web (le serveur Tomcat par exemple), démarrer le serveur Tomcat et lancer votre application.





# Hello World!